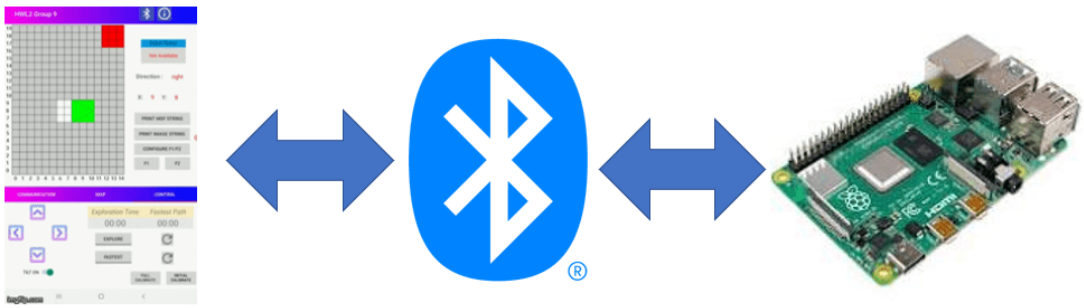
Android

Architecture



The RPi is connected to Nexus 7 tablet via Bluetooth with the use of *rfcomm* protocol. The *robot service* will be the main program where the simulated robot is designed on Android. The grids represent the cells in the arena (unexplored/ explored/ obstacles / free space). The robot and grids would make up the Android program components, making the Android application functional.

Functionalities

1) The Android application (AA) can transmit and receive text strings over the Bluetooth serial communication link
If the *BluetoothConnectionService* == *true*, then the message can be transmitted using the ***MainActivity.printMessage(String Message)*** function. This function will convert the message into bytes and call Bluetooth service class to write over to RPi; BluetoothServices.write(bytes).

2) Functional graphical user interface (GUI) that can initiate the scanning, selection and connection with a Bluetooth device

A *Listview* is a setup for paired devices and newly discovered devices, respectively. The scanning / device discovery is made by calling the ***startDiscovery*** function (requesting for discovery through the *BluetoothAdapter*). When a device is chosen by the user, the **MAC address** of the device is sent back to the parent Activity in the result Intent. A new *ConnectThread* is created to initiate a

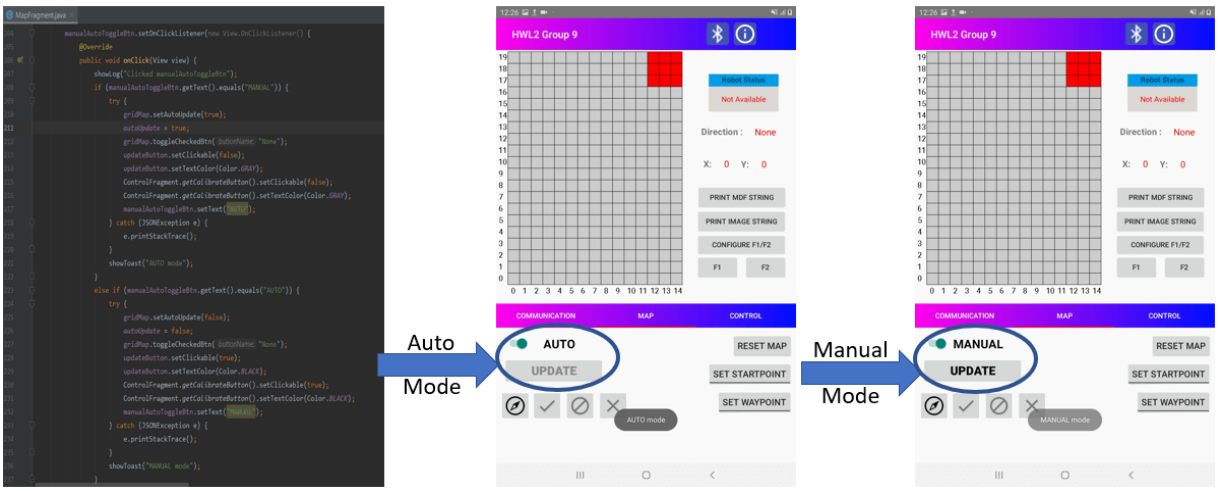
connection with the chosen device. The thread is started by calling ***mConnect-Thread.start()*** in order for the thread to manage the connection and perform transmissions.

3) Functional GUI that provides interactive control of the robot movement via the Bluetooth link (e.g. move forward, left and right)
To control the robot movement, four control buttons, namely, U (Up), R (Right), L (Left) and D (Down), are provided on the main screen of the application. When the user clicks any one of these buttons, a corresponding message is sent to the RPi.

U : *MainActivity.printMessage("A:cmd:forward");*
R : *MainActivity.printMessage("A:cmd:right");*
D : *MainActivity.printMessage("A:cmd:reverse");*
L : *MainActivity.printMessage("A:cmd:left");*

4) Functional GUI that indicates the current status of the robot (e.g. stop, moving, etc.).
When the movement control buttons are pressed, the *printMessage* method will be invoked. The message is saved in *sharedPreferences*. The *Communication-Fragment* class retrieve the string from the *sharedPreferences* and display all the command received and commands to send on the text box at the bottom of the screen.

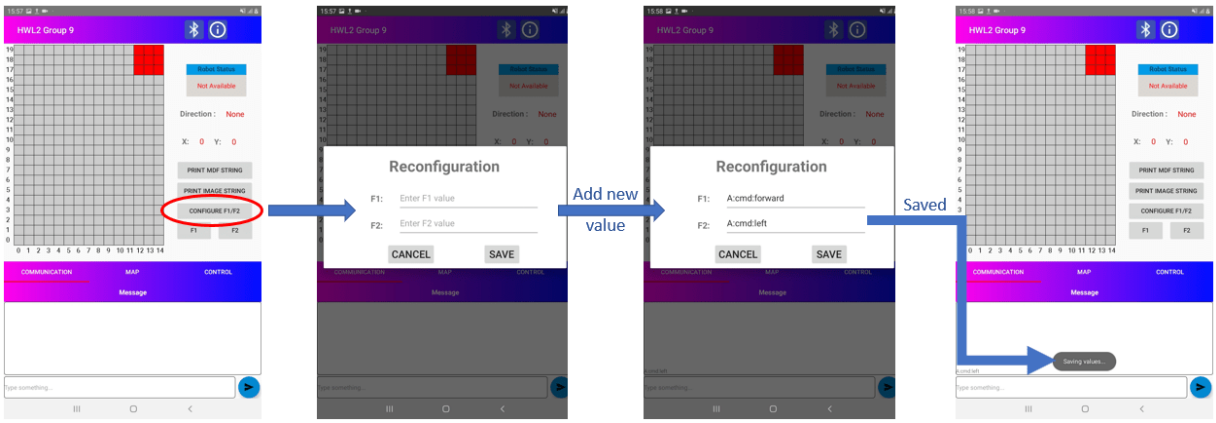
5) Functional GUI that provides the selection of Manual or Auto-updating of the graphical display of the maze environment



The Android GUI allows the selection of Manual or Auto state, through which the grid gets updated. On selecting the auto-update selection the grid automatically gets updated. In the case that the auto-update state is toggled to a manual update, the robot's position on the grid gets updated only when the user manually clicks the movement button. This feature is implemented by calling the

method *manualAutoToggleBtn.setOnClickListener()*, which checks for *manual-AutoToggleBtn* text value (manual or auto) and calls *setMap()* method to update the display on the map.

6) Functional GUI that provides two buttons that supports persistent user re-configurable string commands to the robot



The GUI has two buttons F1 and F2 which contain a string value and are reconfigurable. For implementing the reconfigurable buttons *shared preferences* is used that enables to store the value of the strings and retrieve the values upon restarting the system. On click of either of these buttons the *f1.onClickListener()* or the *f2.onClickListener()* method is called respectively. When the respective method is called the String value that is stored in association with the button is extracted and transmitted to the PC or the robot.

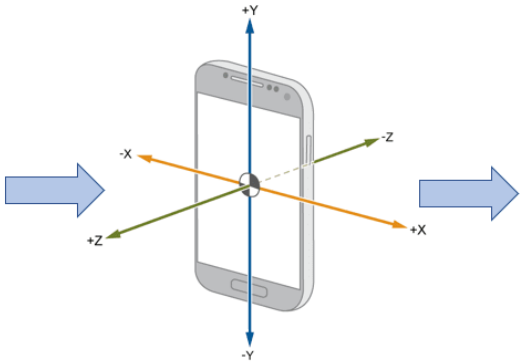
7) Robust connectivity with Bluetooth devices

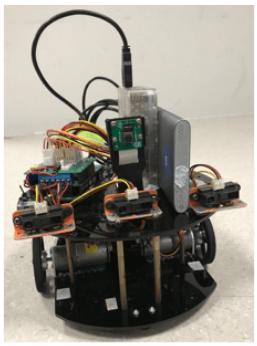
The app allows for scanning for devices and connecting to devices found. Once two devices are connected the connection between them is robust. The Bluetooth connection ensures that even if the connection is lost or gets hung during transmission, the connection can re-establish itself without any extra step from our end.

8) Using tablet IMU to move robot on tilt

```
public void onSensorChanged(SensorEvent event) {
    float x = event.values[0];
    float y = event.values[1];
    float z = event.values[2];
    showLog("SensorChanged X: "+x);
    showLog("SensorChanged Y: "+y);
    showLog("SensorChanged Z: "+z);

    if(sensorFlag) {
        if (y < -2) {
            showLog("Sensor Move Forward Detected");
            gridMap.moveRobot( direction: "forward");
            MainActivity.refreshLabel();
            MainActivity.printMessage("A:cmd:forward");
        } else if (y > 2) {
            showLog("Sensor Move Backward Detected");
            gridMap.moveRobot( direction: "back");
            MainActivity.refreshLabel();
            MainActivity.printMessage("A:cmd:reverse");
        } else if (x > 2) {
            showLog("Sensor Move Left Detected");
            gridMap.moveRobot( direction: "left");
            MainActivity.refreshLabel();
            MainActivity.printMessage("A:cmd:left");
        } else if (x < -2) {
            showLog("Sensor Move Right Detected");
            gridMap.moveRobot( direction: "right");
            MainActivity.refreshLabel();
            MainActivity.printMessage("A:cmd:right");
        }
    }
```





In the application, an addition tilt sensing feature has been implemented using the method *phoneTiltSwitch.setOnCheckedChangeListener()*. Through this feature, the position of the robot can be changed on the grid by tilting the device.
